The dynamic shortest path problem with time-dependent stochastic disruptions

Derya Sever[a], Lei Zhao[b], Nico Dellaert[a], Emrah Demir[c,*], Tom Van Woensel[a], Ton De Kok[a]

[a] *School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, Netherlands*
[b] *Department of Industrial Engineering, Tsinghua University, Beijing, China*
[c] *Panalpina Centre for Manufacturing and Logistics Research, Cardiff Business School, Cardiff University, Cardiff, United Kingdom*

Abstract

The dynamic shortest path problem with time-dependent stochastic disruptions consists of finding a route with a minimum expected travel time from an origin to a destination using both historical and real-time information. The problem is formulated as a discrete time finite horizon Markov decision process and it is solved by a hybrid Approximate Dynamic Programming (ADP) algorithm with a clustering approach using a deterministic lookahead policy and value function approximation. The algorithm is tested on a number of network configurations which represent different network sizes and disruption levels. Computational results reveal that the proposed hybrid ADP algorithm provides high quality solutions with a reduced computational effort.

Keywords: Dynamic shortest path problem, Approximate dynamic programming, Time-dependent disruption, Lookahead policy, Value function approximation

*Corresponding author
Email addresses: derya.sever@gmail.com (Derya Sever), lzhao@tsinghua.edu.cn (Lei Zhao), N.P.Dellaert@tue.nl (Nico Dellaert), demire@cardiff.ac.uk (Emrah Demir), T.v.Woensel@tue.nl (Tom Van Woensel), A.G.d.Kok@tue.nl (Ton De Kok)

1. Introduction

The growth in economic output influences the volume of transportation activities. Within the EU-28 (European Union's 28 countries), about 2,200 billion ton-kilometers of goods were transported in 2014, with road transportation accounting for about three quarters (i.e., 74.9%) of this volume (Eurostat, 2016). The growing volumes of road freight transportation contribute to congestion on road, which leads to delay, disruption, and other negative impacts on the reliability of transportation. The direct impact of congestion, as transportation externality, refers to increased travel times to other entities in the transportation system. Moreover, congestion could indirectly result in increased fuel costs, air pollution, noise pollution and stress levels (Demir et al., 2015).

Congestion can be measured as the sum of recurrent and non-recurrent delay in a traffic network (Skabardonis et al., 2003). The first category depends on the fluctuations in demand and the physical capacity of the road. The second category depends on the nature of the incident, such as breakdowns, and accidents. As the uncertainty in traffic networks increases due to recurrent and non-recurrent delays, a driver needs to take into account the congestion by considering all traffic states that change the travel time with respect to disruption level.

Having real-time traffic information from Intelligent Transportation Systems (ITS) and taking into account the stochastic nature of disruptions can significantly reduce time delays, congestion and air pollution. As a trade-off, for networks with many disruption levels, computing dynamic routing decisions takes very long computational time. In order to improve the speed of computation, real-life applications (e.g., navigation devices) require fast and high quality algorithms as we intended to propose one in this paper. The problem at hand is named as dynamic and stochastic shortest path problem in the literature and has been the topic of extensive research over the last decades (see, e.g., Flatberg et al., 2005; Kumari and Geethanjali, 2010). The existing models in the literature can be categorized into two: Adaptive routing recourse policies (see, e.g., Polychronopoulos and Tsitsiklis, 1996; Fu, 2001) and a Markov decision process (MDP) (see, e.g., Kim et al., 2005a; Thomas and White, 2007; Güner et al., 2012).

In the domain of vehicle routing, the dynamic shortest path problem with anticipation using the MDP is first studied by Kim et al. (2005a). Whenever there is a new information about disruption, their proposed model considers the congestion dissemination and anticipates the route accordingly. For larger networks, as the formulation becomes intractable, Kim et al. (2005b) proposed state space reduction techniques where the authors identified the traffic data that has no added value in the decision making process. In another study, Güner et al. (2012) considered the non-stationary stochastic shortest path problem with both recurrent and non-recurrent congestion using real time traffic information. The authors formulated the problem as the MDP that generates a dynamic routing policy. To prevent the state explosion, the authors limited the formulation to two-links-ahead formulation where they only retrieve the state information for only two links ahead of the current location. At last, Sever et al. (2013) formulated the dynamic shortest path problem with travel time-dependency using the MDP formulation. The authors reduced the computational time by using different levels of real-time and historical information.

For large-scale traffic networks with many disruption levels, obtaining the optimal solution faces the curses of dimensionality, i.e., states, outcomes, and decisions (Powell, 2011). In order to deal with dimensionality problem, we propose Approximate Dynamic Programming (ADP) approach for the investigated problem (see, e.g., Powell et al., 2002; Powell and Van Roy, 2004; Simão et al., 2009). ADP is widely used in other applications and interested readers are referred to Lam et al. (2007); Cai et al. (2009); Medury and Madanat (2013) and Hulshof et al. (2016). The current empirical studies on traffic congestion show that highways can have more than two levels of disruption which are specified according to the speed level and possible spill-back (Helbing et al., 2009; Rehborn et al., 2011). However, increase in disruption levels leads to an exponential state- and outcome-space growth causing the well-known curses of dimensionality. Therefore, it is necessary to propose efficient approximation techniques to deal with many disruption levels in traffic networks.

In this paper, we formulate the dynamic shortest problem with time-dependent stochastic disruptions as the MDP and propose the hybrid ADP algorithm with the clustering approach. The algorithm uses both a deterministic lookahead policy and a value function approximation. To our knowledge, the hybrid ADP algorithm with the clustering approach has not yet been thoroughly investigated for the dynamic shortest path problem. We propose and compare several variations of ADP algorithms for networks with many disruption levels which are similar to real-life traffic situations.

The scientific contribution of this study is twofold: (i) to propose the hybrid ADP with the clustering approach for solving the dynamic shortest path problem in traffic networks with many disruption levels, and (ii) to test various algorithmic variations of the hybrid ADP algorithm. The remainder of this paper is organized as follows. Section 2 presents a mathematical foundation of the investigated problem. Section 3 introduces the MDP model

followed by the introduction of the proposed ADP algorithm. Section 4 discusses the computational results when applying the proposed algorithms to generated instances. Conclusions are stated in Section 5.

2. Mathematical formulation

Consider a traffic network represented by a directed graph consisting of a finite set of nodes and arcs. This network can be represented as $G = (N, A, A_v)$, where $N = \{0, ..., n\}$ is the set of nodes (or intersections), $A = \{(i, j) : i, j \in N \text{ and } i \neq j\}$ is the set of directed arcs, and finally A_v is the set of vulnerable arcs (i.e., $A_v \subseteq A$). The number of vulnerable arcs is defined as $R : R = |A_v|$. Each vulnerable arc, namely $r \in A_v$, can take any value from the disruption level vector, U^r, whose dimension depends on a specific vulnerable arc. The travel time on an arc (i, j) is assumed to follow a discrete distribution function on the positive integers based upon historical data and the disruption level of the arc (i, j). The real-time information about the disruption statuses of all vulnerable arcs is obtained when reached at the next node. The objective of the investigated problem is to minimize the expected travel time between an origin and a destination. The resulting problem is a finite horizon stochastic dynamic optimization problem, which can be formulated as a MDP. In the following subsections, we describe the key elements of the mathematical model, using the notations of Powell (2011).

2.1. State

Stage t represents the number of nodes that have been visited so far from the origin node. The system state S_t at stage t is represented by the following two components, $S_t = (i_t, \hat{D}_t)$, $t = 0, ..., T$:

i_t	The current node at stage t ($i_t \in N$)
$\hat{D}_t$	The disruption status vector which gives disruption statuses of all vulnerable arcs at stage t.

The terminal stage T can be reached by arriving at the destination. Note that, at each stage, a realization of the disruption vector $\hat{D}_t$ is used. For each vulnerable arc, there can be K_r different types of disruption levels: $\hat{D}_t(r) \in U^r$: $U^r = \{u^1, u^2, ...u^{K_r}\}, \forall r \in A_v$.

Moreover, the initial state is defined as $S_0 = (\text{origin node}, \hat{D}_0)$, and the final goal state is defined as $S_T = (\text{destination node}, \hat{D}_T)$, where $\hat{D}_0$ and $\hat{D}_T$ are the realizations of the disruptions for all vulnerable arcs at the initial and final stage, respectively. We note that the goal state is absorbing and cost free.

2.2. Decision variable (action)

At the beginning of each stage t, a decision (or action) is to made based on the system state $S_t = (i_t, \hat{D}_t)$. The decision variable x_t shows the next node to visit for a given state S_t. We note that each action (i.e., $x_t = i_{t+1}$) is an element of the set of all possible actions (i.e., $x_t \in X^\pi(S_t)$). The decision policy can be described as

$X^\pi(S_t)$	Decision function that determines the move decision x_t at stage t under policy π for a given state S_t
Π	Set of possible policies. Each element $\pi \in \Pi$ corresponds to a different policy.

2.3. Exogenous information process

The disruption statuses of the vulnerable arcs may change as a vehicle proceeds to the next stage $t + 1$. The exogenous information consists of the realization of the disruption statuses of all the vulnerable arcs. The system's exogenous information W_{t+1} at the next stage $t + 1$ can be stated as

$$\hat{D}_{t+1} = W_{t+1}, \tag{1}$$

where $\hat{D}_{t+1}$ denotes the disruption status realization that becomes known between stages t and $t + 1$.

2.4. Cost function

The cost function associated with the system state and decision variable can be calculated as the travel time from the current node, i_t, to the next node, $x_t = i_{t+1}$ for a given realized disruption status. The cost function can be formulated as

$$C(S_t, x_t) \quad = \quad t_{i_t, x_t}(\hat{D}_t), \tag{2}$$

where t_{i_t, x_t} is the travel time of the current arc for a given realized disruption status.

2.5. Transition function

The transition function depicts the system state transition. At stage t, a decision is made and then the exogenous information is observed. The system transits to a new state S_{t+1} according to the transition function: $S_{t+1}= S^M(S_t, x_t, W_{t+1})$. Note that "M" represents model as in Powell (2011). The state transition involves the following transition functions:

$$i_{t+1} = x_t, \tag{3}$$

$$D_{t+1} = \hat{D}_{t+1}. \tag{4}$$

The disruption status vector transits from $\hat{D}_t$ to $\hat{D}_{t+1}$ according to a Markovian transition matrix. We note that D_{t+1} is the vector of random variables representing the disruption status of each vulnerable arc in the network for the next stage. We define the transition matrix of a vulnerable arc r from stage t to $t+1$ as $\Theta^r(t|S_t, x_t)$. This transition matrix is dependent on the state and the travel time of the current arc. The probability of being in the disruption status of the next stage $\hat{D}_{t+1}$ depends on the travel time between i_t and i_{t+1} given the disruption status realization.

We note that the travel time given disruption status is a positive integer. Let $p^r_{u,u'}$ denote the unit-time transition probability between any two disruption levels of the vulnerable arc r, $p^r_{u,u'} = P\{\hat{D}_{t+1}(r) = u' | \hat{D}_t(r) = u\}$. $\Theta^r(t|S_t, x_t)$ is the transition matrix of the vulnerable arc r considering the travel-time-dependency given the current travel time based on the disruption status realization, $\hat{D}_t$:

$$\Theta^r(t|S_t, x_t) = \begin{bmatrix} p^r_{u^1,u^1} & p^r_{u^1,u^2} & \cdots & p^r_{u^1,u^{K_r}} \\ p^r_{u^2,u^1} & p^r_{u^2,u^2} & \cdots & p^r_{u^2,u^{K_r}} \\ . & & & \\ . & & & \\ p^r_{u^{K_r},u^1} & p^r_{u^{K_r},u^2} & \cdots & p^r_{u^{K_r},u^{K_r}} \end{bmatrix}^{t_{i_t,x_t}(\hat{D}_t)} \tag{5}$$

The probability of having the new disruption status $\hat{D}_{t+1}$ for a given $\hat{D}_t$ is then calculated as $(\Theta^r_{u,u'}(t|S_t, x_t)$, which indicates the specific row and column index in the matrix $\Theta^r(t|S_t, x_t))$. The probability can be calculated as

$$P(\hat{D}_{t+1}|\hat{D}_t) = \prod_{r=1}^{R} \Theta^r_{u,u'}(t|S_t, x_t), \tag{6}$$

where r is the counter for vulnerable arc(s), and R is the maximum number of vulnerable arc.

It is noted that we have autocorrelation for the travel time on the arc, but we do not have correlation between travel times on neighboring links. Our model and approach will also work in a correlated situation; non-correlation is not an assumption in that part.

2.6. Objective function

The objective is to minimize the expected total travel time from the origin node to the destination node and can be calculated as

$$\min_{\pi \in \Pi} \quad \mathbb{E}(\sum_{t=1}^{T} C(S_t, X^\pi(S_t))), \tag{7}$$

where $\mathbb{E}(..)$ represents the total expected cost of travel time from the beginning to the end of horizon T.

3. The approximate dynamic programming approach

In this section, we first discuss some key algorithmic issues that we have encountered in the design of ADP algorithms. Then, we present several ADP algorithms specifically developed for the investigated problem.

If we discretize the continuous state space and transition function, the problem described in the previous section becomes a discrete state MDP with a stochastic finite planning horizon. The optimization problem in

equation (7) can be solved using the Bellman's equation:

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} C(S_t, x_t) + \sum_{S_{t+1}} P(S_{t+1}|S_t) V_{t+1}(S_{t+1}), \tag{8}$$

$$V_T(S_T) = 0. \tag{9}$$

where $V_t(S_t)$ is the value of being in state S_t, and $V_T(S_T)$ is the value of being in stage T. Moreover, the decision becomes the following:

$$x_t^* = \arg\min_{x_t \in \mathcal{X}_t} C(S_t, x_t) + \mathbb{E}(V_{t+1}(S_{t+1})). \tag{10}$$

The MDP faces the curses of dimensionality (i.e., states, outcomes, and decisions) with the increase in the number of disruption levels and the number of nodes. To solve large-scale problems with many disruption levels, the backward dynamic programming approach for solving the Bellman's equations becomes computationally intractable. In the literature, several techniques such as state-reduction techniques (Kim et al., 2005b; Thomas and White, 2007) and stochastic lookahead algorithms (Güner et al., 2012; Sever et al., 2013) are used to solve MDP problems with reduced computational time. As an alternative to these two approaches, ADP can be used as a powerful tool to overcome the curses of dimensionality, especially for complex and large scale problems as shown in Powell (2011).

In this paper, we use the ADP approach with a value iteration. The essence of this approach is to replace the actual value function $V_t(S_t)$ with an approximation value function $\bar{V}_t(S_t)$. ADP proceeds by estimating the approximate value $\bar{V}_t(S_t)$ iteratively. For the investigated problem, this means that arcs are repeatedly traversed to estimate the value of being in a specific state. Let $\bar{V}_{t+1}^{n-1}(S_{t+1})$ be the value function approximation after $n-1$ iterations. Instead of working backward through time as in the traditional DP approach, ADP works forward in time.

The optimization problem using the ADP approach can be rewritten as

$$\hat{v}_t^n = \arg\min_{x_t \in X_t} C(S_t^n, x_t^n) + \sum_{S_{t+1}} P(S_{t+1}|S_t) \bar{V}_{t+1}^{n-1}(S_{t+1}). \tag{11}$$

3.1. Key algorithmic issues in ADP

This section discusses the key algorithmic issues in the design of ADP algorithms to improve the solution time while effectively solving the dynamic shortest path problem.

3.1.1. Post-decision state

The approximate value of being in state S_t^n at iteration n, $\bar{V}_t^n(S_t^n)$, contains the expectation over all possible states at the next stage. We adopt the post-decision state variable as suggested by Powell (2011). We define the post-decision state as S_t^x. It represents the state immediately after a decision at the current stage is made.

$$S_t^x = S^{M,x}(S_t, x_t) = (x_t, \hat{D}_t) = (i_{t+1}, \hat{D}_t). \tag{12}$$

The post-decision state eliminates the expectation calculation in Bellman's equation by using the deterministic value of choosing an action given the current state S_t.

3.1.2. Value function approximation with the lookup table representation

We use the lookup table representation for the value function approximation. This means that for each discrete state S_t, an estimate $\bar{V}_t(S_t)$ is calculated. As we use a post-decision state, the value of being in the post-decision state of the previous stage should be updated using $\hat{v}_t^n$:

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1-\alpha_{n-1}) \bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1} \hat{v}_t^n. \tag{13}$$

The equation (13) updates the state value using a weighted average of the current estimate and the estimate from the previous iteration.

3.1.3. Exploration and exploitation strategies

In the ADP algorithm, a value function is estimated by visiting states as the value of being in a certain state. If an exploration strategy is used, states are visited to improve the estimates of the value of being in the state regardless the decision gives the best value. However, if an exploitation strategy is applied, a decision that gives the best value is performed. One of the challenges in ADP is to ensure a right balance between exploration and exploitation strategies when making a decision for a given certain state. Using only exploitation strategy may lead to stuck into a local optima by choosing the decision that appears to have the best value. On the other hand, using only exploration strategy may lead to visit every state randomly, which may significantly increase the computational time while leading to poor value estimates.

In the early iterations of the algorithm, the value of being in a state is highly dependent on the sample path and the initial solution. Therefore, the exploration strategy should be used more often in the early iterations to improve the quality of state values. For this purpose, we use a mixed of strategies in our algorithms as in Powell (2011). To do this, a random probability of choosing the best action or choosing alternative actions is applied. We ensure that the probability of choosing the best action increases as the state is visited more often. We use exploration rate, ρ_t, for choosing the next best alternative as $\rho_t = b/n'(S_t)$. We should note that the exploration rate decreases with the number of visits to the particular state, $n'(S_t)$, and increases with the user-defined parameter b.

3.1.4. Initialization heuristics

Initial values play an important role in the accuracy of the approximate values and the computational performance of the ADP algorithm. For this purpose, we investigate the effect of two initialization heuristics: A deterministic approach and a stochastic two-arc-ahead policy with memoryless probability transitions $DP(2, M)$.

In the deterministic approach, initial state values are determined by solving a deterministic shortest path problem, assuming that the probability of having a disruption is zero. In determining a route, only non-disruption state, for all vulnerable arcs, is considered. In the transition probabilities of all vulnerable arcs, the non-disruption state becomes an absorbing state in equation (5). Then, equations (8) - (10) are solved. Note that the output of this initialization heuristic is independent of the disruption state.

As an alternative initialization heuristic, we applied the $DP(2, M)$ heuristic as proposed in Sever et al. (2013). In this heuristic, we have a real-time information of only two-arc-ahead from the current node. Therefore, the state space of the current node i_t is modified as $S_t = (i_t, \hat{D}_t^{i_t})$, where the disruption status of the current node is $\hat{D}_t^{i_t} = \{u_t^{r_{i_t 1}}, u_t^{r_{i_t 2}}, .., u_t^{r_{i_t R_{i_t}}}\}$. Moreover, r_{i_t} is the vector of the vulnerable arcs that are in the two-arc-ahead neighborhood of node i_t and $R_{i_t} = |r_{i_t}|$, $R_{i_t} \leqslant R$. For the rest of the arcs, expected travel times are calculated. The transition probability vector in equation (5) is no longer travel time-dependent and only a part of the transition matrix with only the vulnerable arcs in the two-arc-ahead neighborhood is considered. We note that the output from this initialization heuristic is dependent on the disruption state.

3.2. The generic ADP algorithm

In the generic ADP, we adopt the value function approximation algorithm with a post-decision state variable. We use both exploration and exploitation strategies to update value function approximations. We now present two types of ADP algorithm (i.e., single-pass and double-pass).

In the ADP algorithm with single-pass, updating the value function takes place as the algorithm progresses forward in time. According to the Algorithm 1, in Step 2a, at stage t, an updated estimate of the state value of being in state S_t is obtained. In Step 2b, the estimate of $\bar{V}_{t-1}^n$ is updated by using the state value from the previous iteration, $\bar{V}_{t-1}^{n-1}$, and the current value estimate $\hat{v}_t^n$. Note that information is collected as the algorithm proceeds in time due to the fact that the disruption transition rates for the future stage is dependent on the current travel time.

As an alternative method to the single-pass value iteration algorithm, a double-pass algorithm is also introduced in Algorithm 2. This algorithm first steps forward in time by creating a trajectory of states, actions and outcomes in Step 2. Then, the value function is updated by stepping backwards through the stages in Step 3. An update of the value function is conditional on whether exploitation or exploration strategy is used. If it is an exploitation strategy at stage t, then the value function of the state at stage t is updated. If it is an exploration strategy at stage t, the value function is updated only if the explored action improves the value function compared to the exploitation in Step 3b.

3.3. The hybrid ADP with the clusters

This section introduces the hybrid ADP with the clustering approach by using the deterministic lookahead policy and the value function approximations. The state variable in our MDP formulation is a representation of

Algorithm 1: ADP algorithm with single-pass

Step 0: Initialization
Step 0a: Initialize $\bar{V}_t^0$, $\forall t$ by using the value from the initialization heuristic.
Step 0b: Set n=1.
Step 1: Choose a sample disruption vector with Monte Carlo simulation for n and for $t = 0$, and initialize $S_0^n = (\text{origin node}, \hat{D}_0^n)$.
Step 2: Do for all $t = 0, ..., T-1$ (where T is reached by arriving at the destination node)
Step 2a: Choose a random probability p and exploration rate ρ_t as $\rho_t = 0.2/n'(S_t)$

$$\hat{v}_t^n = \min_{x_t \in N, x_t \in X_t} \quad C(S_t^n, x_t^n) + \bar{V}_t^{n-1}(x_t, \hat{D}_t^n). \tag{14}$$

The node x_t that gives the optimal value is denoted as x_t^{*n}.

$$\hat{v}_t^n = \begin{cases} \text{solve Equation (14) for } x_t \in N & \text{if} \quad p \geqslant \rho_t, \\ \text{solve Equation (14) for } x_t \in N \backslash x_t^{*n} & o.w. \end{cases}$$

The node that is chosen is denoted as x_t^n and becomes the next node to visit.
Step 2b: If $t > 0$, update $\bar{V}_{t-1}^n(S_{t-1}^{x,n})$ using:

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n.$$

We use the harmonic stepsize: $\alpha_{n-1} = \frac{a}{a+n'(S_t)-1}$ and $n'(S_t)$ is the total number of visits to state S_t.
Step 2c: Find the post-decision state: $S_t^{x,n} = (i_{t+1}, \hat{D}_t^n)$.
Step 2d: Find the next pre-decision state: $S_{t+1}^n = (i_{t+1}, W_{t+1}^n)$.
Step 3: Increment n. If $n \leqslant \mathbb{N}$ go to Step 1. Note that $\mathbb{N}$ denotes the pre-set maximum number of iterations.
Step 4: Return the value functions $(\bar{V}_t^{\mathbb{N}})_{t=1}^T$.

Algorithm 2: ADP algorithm with double-pass

Step 0: Initialization
Step 0a: Initialize $\bar{V}_t^0$, $\forall t$ by using the value from the initialization heuristic.
Step 0b: Set n=1.
Step 1: Choose a sample disruption vector with Monte Carlo simulation for n and for $t = 0$, and initialize $S_0^n = (\text{origin node}, \hat{D}_0^n)$.
Step 2: (Forward pass) Do for all $t = 0, 1, ..., T-1$ (where T is reached by arriving at the destination node)
Step 2a: Solve:

$$\hat{v}_t^n = \min_{x_t \in N, x_t \in X_t} \quad C(S_t^n, x_t^n) + \bar{V}_t^{n-1}(x_t, \hat{D}_t^n). \tag{15}$$

The node x_t that gives the optimal value is denoted as x_t^{*n}.

$$\hat{v}_t^n = \begin{cases} \text{solve Equation (15) for } x_t \in N & \text{if} \quad p \geqslant \rho_t, \\ \text{solve Equation (15) for } x_t \in N \backslash x_t^{*n} & o.w. \end{cases}$$

The node x_t that gives the optimal value is denoted as x_t^{*n} and $x_t^{\Delta n}$ if it is from the exploration.
Step 2b: If we choose exploration with $x_t^{\Delta n}$ compute:

$$\bar{V}_{t-1}^{*n}(S_{t-1}^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^{*n},$$
$$\hat{v}_t^{*n} = C(S_t^n, x_t^{*n}) + \bar{V}_t^{n-1}(x_t^{*n}, \hat{D}_t^n).$$

Step 2c: Find the post-decision state: $S_t^{x,n} = (i_{t+1}, \hat{D}_t^n)$.
Step 2d: Find the next pre-decision state: $S_{t+1}^n = (i_{t+1}, W_{t+1}^n)$.
Step 3: (Backward pass) Do for all $t = T-1, ..., 1$
Step 3a: Compute $\hat{v}_t^n$ using the decision x_t^n from the forward pass:

$$\hat{v}_t^n = C(S_t^n, x_t^n) + \hat{v}_{t+1}^n. \tag{16}$$

Step 3b: If $t > 1$, update $\bar{V}_{t-1}^n(S_{t-1}^{x,n})$ using:

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = \begin{cases} (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n & \text{if} \quad x_t^{*n}, \\ (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n & \text{if} \quad x_t^{\Delta n} \,\&\, \quad \bar{V}_{t-1}^{\Delta n}(S_{t-1}^{x,n}) < \bar{V}^{*n}_{t-1}(S_{t-1}^{x,n}). \end{cases}$$

We compute $\bar{V}_{t-1}^{\Delta n}(S_{t-1}^{x,n})$ if we have explored at stage t with action $x_t^{\Delta n}$:

$$\bar{V}_{t-1}^{\Delta n}(S_{t-1}^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n,$$

where the harmonic stepsize is $\alpha_{n-1} = \frac{a}{a+n'(S_t)-1}$ and $n'(S_t)$ is the number of visits to the current state.
Step 4: Increment n. If $n \leqslant \mathbb{N}$ go to Step 1. Note that $\mathbb{N}$ denotes the pre-set maximum number of iterations.
Step 5: Return the value functions $(\bar{V}_t^{\mathbb{N}})_{t=1}^T$.

the nodes and the disruption statuses of the vulnerable arcs. When the network size and the number of disruption levels increase, the number of states and estimated state values in the lookup table also increase. To reduce the computational time while maintaining good solution quality for large size instances, we exploit the structure of the disruption transition function in equation (5) which is travel time-dependent. The structure of the disruption transition function shows within travel time between two nodes, the disruption statuses of the vulnerable arcs at the next stage remain similar to the disruption statuses at the current stage. In other words, in a close neighborhood of the current node, i_t, the disruption statuses of the observed vulnerable arcs at stage t do not change dramatically

as compared to those at stage $t-1$. This structure is a good motivation to cluster the nodes that are close to each other. Using the clustering idea, we introduce a hybrid ADP algorithm with clusters. In this hybrid algorithm, we first apply a deterministic lookahead policy within the cluster and estimate the cost outside the cluster until the destination using value function approximations. Fig. 1 illustrates how a cluster is formed and the hybrid ADP algorithm is performed.

"PLACE Fig. 1 ABOUT HERE"

The cluster of the current node i_t is formed by simply considering the nodes that are two-arc-ahead from the node i_t and denoted as CL_{i_t}. It is assumed that the disruption status $\hat{D}_t$ stays the same within the cluster and a deterministic lookahead policy within the cluster is applied. The lookahead policy consists of solving Dijkstra's algorithm (Dijkstra 1959) within the cluster. In this way, a shortest path from i_t until the next node $i_{t+1}^{CL_{i_t}}$ in the cluster CL_{i_t} is obtained. The cost is determined by Dijkstra's shortest path algorithm and is denoted as $\tilde{C}_t(i_t, i_{t+1}^{CL_{i_t}}, \hat{D}_t)$. Then, the cost of outside the cluster from the node $i_{t+1}^{CL_{i_t}}$ is estimated until the destination using value function approximations. The next node to visit is found with the exploration and exploitation strategies as shown in Algorithms 1 and 2. The current state value, $\bar{V}_t(S_t)$, is updated using the harmonic stepsize rule. Finally, the next cluster consisting of the two-arc-ahead nodes of the next node is found and its value is updated. The algorithm always go further towards the destination node.

3.3.1. *Algorithmic variations in the hybrid ADP with clusters*

This section discusses several algorithmic variations.

- *The size of the cluster*: Three different sizes are considered: (i) one-arc-ahead neighborhood, (ii) two-arc-ahead neighborhood, and (iii) three-arc-ahead neighborhood of the current node i_t. These variations are represented as CL_x, where $x \in \{1,2,3\}$. The deterministic lookahead policy is applied to obtain the cost $\tilde{C}_t(i_t, i_{t+1}^{CL_{i_t}}, \hat{D}_t^n)$ between the current node i_t and the next node $i_{t+1}^{CL_{i_t}}$, where $i_{t+1}^{CL_{i_t}}$ is an element from these clusters. Note that the hybrid ADP algorithms with clusters considering one-arc-ahead neighborhood is equivalent to the standard ADP algorithm as presented in Algorithms 1 and 2.

- *The update of the state values in a shortest path within the cluster*: Another variation of the hybrid algorithm is to decide whether to update the state values of the intermediate nodes on the shortest path. For each shortest path consisting of nodes to travel until $i_{t+1}^{CL_{i_t}}$, we investigate whether updating the state values of the path obtained from the lookahead policy improves our solution or not. For instance, the current node is i_t and $i_{t+1}^{CL_{i_t}}$ is the next node by using the shortest path $i_t \rightarrow z \rightarrow i_{t+1}^{CL_{i_t}}$. If the state value of node i_t is updated but z is not updated, we call this as "No-Update" NU. If the state value of node z is also updated, then we call this as "Update" U. Note that in the hybrid ADP algorithms with the cluster size of one-arc-ahead neighborhood, CL_1, the shortest path consists of current node and decision node so the algorithmic design variation for "Update" has no added value. Therefore, for consistency with other hybrid ADP algorithms, CL_1 is denoted with "No-Update".

- *The update of the state values*: We study both single-pass and double-pass approaches to update the state values. For the single-pass algorithm with CL_2 and CL_3 approach with or without updating the values of the nodes on shortest path within the cluster ($CL_{(x,S,U)}$, $CL_{(x,S,NU)}$ and $x \in \{2,3\}$), we use Algorithm 1. Similarly, for the double-pass algorithm with CL_2 and CL_3 approach ($CL_{(x,D,U)}$, $CL_{(x,D,NU)}$), we use Algorithm 2. For both algorithms, the value function in Equations (14) and (15) becomes:

$$\hat{v}_t^{*n} = \min_{i_{t+1}^{CL_{i_t}}} \quad \tilde{C}_t(i_t, i_{t+1}^{CL_{i_t}}, \hat{D}_t^n) + \bar{V}_t^{n-1}(i_{t+1}^{CL_{i_t}}, \hat{D}_t^n).$$

3.3.2. *A benchmark heuristic*

For large instances, where the optimal solutions are not obtainable, we benchmark with the stochastic two-arc-ahead policy $DP(2,H)$ introduced in Sever et al. (2013), which is shown to perform only slightly worse than the optimal algorithm.

In this policy, online information of the arcs that are two-arc-ahead from the current node is retrieved. Therefore, the state space of the hybrid policy for any current node i_t is modified as $S_t = (i_t, \hat{D}_t^i)$, where $\hat{D}_t^i = \{u_t^{r_{i_t1}}, u_t^{r_{i_t2}}, .., u_t^{r_{i_tR_{i_t}}}\}$ only include the vector of the vulnerable arcs that are in the two-arc-ahead neighborhood of node i_t and $R_{i_t} = |r_{i_t}|$.

In this heuristic, we use the same travel time-dependent transition matrix in equation (5) except we only consider the limited part of the transition matrix. For the vulnerable arcs outside the neighborhood, the time-invariant probability distributions is calculated. We solve equations (8) – (10) using a backward recursion algorithm given in Sever et al. (2013).

4. Computational results

This section presents results of extensive computational experiments performed to assess the performance of three algorithms; (i) the hybrid ADP with clusters, (ii) the dynamic programming with stochastic two-arc-ahead policy $DP(2,H)$, and (iii) the optimal MDP. We first describe our test instances and evaluation methods in Section 4.1. In Section 4.2, we analyze several algorithmic variations. Finally, we compare the performance of the proposed ADP algorithm with the $DP(2,H)$ and the MDP algorithm in Section 4.3.

Throughout the experiments, we fix the parameters in the ADP algorithms based on our preliminary test results. We set the constant in the harmonic stepsize rule to 5. We set the exploration rate as $0.2/n'(S_t)$, where $n'(S_t)$ is the number of visits to state S_t. Moreover, we fix the total number of ADP iterations to $\mathbb{N} = 100,000$.

4.1. Data and experimental setting

The algorithms presented in this paper are implemented in Java. All experiments are conducted on a personal computer with an IntelCore Duo 2.8 Ghz Processor and 3.46 GB RAM. We generated our test instances based on the following network properties.

Network size: The size of the networks ranges from 16 to 100 nodes. Each instance is designed such that the origin-destination pairs are located from the top-left to the bottom-right corners.

Network vulnerability: The network vulnerability is defined as the percentage of vulnerable arcs in the network. Instances with 50% and 80% of vulnerable arcs are considered as low and high network vulnerability, respectively.

Number of disruption levels: The vulnerable arcs in a network include two, three or five disruption levels. Note that the disruption levels also include the non-disrupted case where there is no disruption at all. For each vulnerable arc the possible disruption levels are also kept the same.

Disruption rate: Disruption rate is defined by the steady state probability of having a disruption on a vulnerable arc. A low probability of having disruptions is set to be in between $[0.1 - 0.5)$ and a high probability is set to be in between $[0.5 - 0.9)$.

Travel time: Travel time of each arc for the non-disrupted level is randomly selected beforehand from a discrete uniform distribution $U(1,10)$. If there is a disruption, travel time for the non-disrupted level is multiplied by a scalar depending on the disruption level.

We generated 48 different instance types with properties as shown in Table 1. For each instance type, we randomly generate 50 replications and in total 2,400 test instances.

"PLACE TABLE 1 ABOUT HERE"

Depending on the instance size, we use either exact evaluation or evaluation via simulation. When the optimal MDP algorithm is computationally tractable within the one-hour time limit, we perform an exact evaluation. For each algorithm, an exact value function of the resulting policy is computed by enumerating for all states via a backward recursion. For test instances where the optimal MDP algorithm is not tractable within the one-hour time limit, we evaluate the algorithm via simulation. For each instance, we simulate with a sample of size 5,000 and compute the sample averages as the evaluated cost.

4.2. Algorithmic variations of the hybrid ADP algorithm with clusters

We now compare four algorithmic variations of the hybrid ADP algorithm with clusters; (i) initialization heuristic, (ii) state value update with single-pass or double-pass, (iii) update or no-update of the state values on the shortest path within the cluster, and (iv) cluster size. A summary of algorithmic variations are given in Table 2. For presentation convenience, we use $C[Algorithm]$ to denote the evaluated cost (or expected total travel time) of the corresponding algorithmic design.

"PLACE TABLE 2 ABOUT HERE"

4.2.1. An analysis of initialization heuristics

We first compare two heuristics for obtaining initial value function estimates: A deterministic approach, denoted as $INIT_D$, and a stochastic two-arc-ahead policy with memoryless probability transitions $DP(2, M)$, denoted as $INIT_S$. To analyze whether there is a significant performance difference between these two initialization heuristics, we applied a paired t-test with significance level of 0.05 as shown in Table 8 in the Appendix. In Fig. 2.a/b, we report the mean and 95% confidence interval of the cost differences $\Delta_I = C[INIT_D] - C[INIT_S]$. Moreover, we present the results with number of disruption levels $K = 2, 3$ and network sizes $N = 16, 64$.

"PLACE Fig. 2 ABOUT HERE"

Fig. 2 shows that the $DP(2, M)$ initialization heuristic significantly outperforms the deterministic initialization heuristic in most of the instances. Moreover, the significance increases with the number of disruption level and network size. Therefore, in the rest of our experiments, we adopt the $DP(2, M)$ as the initialization heuristic.

4.2.2. An update of the state value

In Table 3, we compare the single-pass and double-pass state value update approaches. The percentage cost improvement is calculated as $\Delta_{SD}(\%) = \frac{C[\text{Single-pass}] - C[\text{Double-pass}]}{C[\text{Single-pass}]} * 100$. For each combination of network size and number of disruption level, we report the minimum, maximum, and mean of Δ_{SD} over all test instances.

"PLACE TABLE 3 ABOUT HERE"

Table 3 shows that, on average, the double-pass approach slightly outperforms the single-pass approach with cluster sizes CL_2 and CL_3. Table 9 in the Appendix shows the significance of the main effect. Furthermore, the performance gap between single- and double-pass approaches reduces as the number of disruption levels and the network size increase. However, the relative performance between single- and double-pass approaches remains unclear for cluster size CL_1. While we tentatively select the double-pass approach for cluster sizes CL_2 and CL_3, we investigate both single- and double- approaches in the following experiments.

4.2.3. An update of the nodes on the shortest path

In the hybrid ADP algorithm with clusters, besides updating the state value of the node at the end of the cluster, we can decide on whether to update the state values of the intermediate nodes on the shortest path within the cluster. In Table 4, we present the comparison results between the above two approaches. The percentage cost improvement is calculated as $\Delta_{UN}(\%) = \frac{C[\text{No-update}] - C[\text{Update}]}{C[\text{No-update}]} * 100$. For each combination of network size and number of disruption levels, we report the minimum, maximum, and mean of Δ_{UN} over all test instances.

"PLACE TABLE 4 ABOUT HERE"

Table 4 clearly shows that the update approach outperforms the no-update approach when we use the double-pass state value update and we observe the opposite effect when we use the single-pass state value update. Table 9 in the Appendix indicates significant interaction between the single- versus double-pass and the update versus no-update. We note that the update approach outperforms the no-update approach especially under the combination of double-pass and cluster size CL_2. This effect decreases with the number of disruption levels and increases with network size. In the following experiments on cluster sizes, we investigate the combination of double-pass with update approach.

4.2.4. The size of a cluster

In Table 5, we present the comparison results of three cluster sizes; (i) one-arc-ahead CL_1, (ii) two-arc-ahead CL_2, and (iii) three-arc-ahead CL_3 neighborhood. The percentage cost improvement Δ_{CS} is calculated relative to CL_1. For example, in row $CL_{(1-2,D,NU)}$, we compute $\Delta_{CS}(\%) = \frac{C[CL_1] - C[CL_2]}{C[CL_1]} * 100$, where we use double-pass and update of state values for the nodes on the shortest path within the cluster. Similar to the previous tables, we report the minimum, maximum, and mean of Δ_{CS} over all test instances.

"PLACE TABLE 5 ABOUT HERE"

Table 5 indicates that both CL_2 and CL_3 outperform CL_1. And, the improvement increases with the number of disruption levels and network size. On the contrary, we observe that CL_2 outperforms CL_3 in most of the cases in Table 5. Table 9 also indicates the significance of the main effect of cluster size.

Based on our numerical experiments on the design variations of the hybrid ADP algorithm, we select the $CL_{(2,D,U)}$ with value update of nodes on the shortest path within the cluster and double-pass approach for state value update.

4.3. A comparison of the algorithms

In this section, we analyze the $CL_{(2,D,U)}$, the MDP, and the $DP(2,H)$. The comparison is done with respect to their estimated total cost defined by the total travel time computed according to the relevant evaluation method. We also provide the percentage gap of each algorithm with respect to the $DP(2,H)$. We show the average results over all 2,400 test instances depending on the network size, the disruption rate and the network vulnerability.

Tables 6 and 7 provide a detailed analysis on the results. The first column of each table indicates the network size. The second column states the disruption rate. The column "Performance" presents the total cost, the standard deviation of the cost values and the percentage GAP with respect to the $DP(2,H)$. For each disruption level, we also present the cost values obtained with three different algorithms. Note that negative sign in the percentage gap means that the algorithm outperforms the $DP(2,H)$.

4.3.1. The $CL_{(2,D,U)}$ versus the $DP(2,H)$

Tables 6 and 7 show that the hybrid ADP algorithm with clusters considering two-arc-ahead neighborhood $CL_{(2,D,U)}$ performs within -2.64% and 0.96% away from the $DP(2,H)$. As the network size becomes larger, the significant difference between the $CL_{(2,D,U)}$ and the $DP(2,H)$ decreases such that in large size networks the solution quality of the $CL_{(2,D,U)}$ is higher than or equal to the $DP(2,H)$. For instance, in networks with low network vulnerability and low disruption rate, the performance of the $CL_{(2,D,U)}$ relative to the $DP(2,H)$ increases from small to large networks by the following gap percentages: 0.43% to -0.48%, 0.41% to -0.26% and 0.22% to -0.94% for 2, 3 and 5 disruption levels, respectively.

Moreover, as network vulnerability becomes higher, the performance of the $CL_{(2,D,U)}$ relative to the $DP(2,H)$ increases. For large and highly vulnerable networks, the $CL_{(2,D,U)}$ outperforms $DP(2,H)$ for all disruption levels and disruption rates. For large networks with low disruption rate, the performance increase of the $CL_{(2,D,U)}$ relative to the $DP(2,H)$ is even more significant as vulnerability increases from low to high: 0.33% to -2.64%, 0.39% to -1.01% and 0.96% to -1.00% for 2, 3 and 5 disruption levels, respectively.

In the networks with high disruption rates, in general the performance gap between the $CL_{(2,D,U)}$ and the $DP(2,H)$ diminishes when compared to the networks with low disruption rates. For instance, in Table 6, in small networks with 2-disruption levels, the $DP(2,H)$ outperforms the $CL_{(2,D,U)}$ by 0.43% (with significant difference) in low disruption rate and by 0.21% (with no significant difference) in high disruption rate. This is because as disruption rate increases, routing algorithms tend to make similar routing decisions considering more risk averse arcs.

"PLACE TABLE 6 ABOUT HERE"
"PLACE TABLE 7 ABOUT HERE"

4.3.2. The $CL_{(2,D,U)}$ versus the optimal MDP

The comparison between the $CL_{(2,D,U)}$ and the optimal MDP is limited to the small and medium networks with 2-disruption levels where we can compute the optimal policy. Tables 6 and 7 present that as network size and disruption rate increase, the performance gap between the $CL_{(2,D,U)}$ and the MDP decreases. For example, in low network vulnerability and low disruption rate, the performance gap decreases from 0.72% to 0.48% for small and large networks, respectively. On the other hand, as the network vulnerability becomes higher, the performance gap between the $CL_{(2,D,U)}$ and the MDP increases. For instance, in small networks with low disruption rate, the performance gap increases from 0.72% to 1.04% for low and high network vulnerability, respectively.

4.3.3. An analysis of computational time

Fig. 3 shows the computational time of the $CL_{(2,D,U)}$ and the $DP(2,H)$ with respect to different network sizes for a given disruption level. When the network size increases from small to large with 2-disruption levels, Fig. 3-(a) shows that the $CL_{(2,D,U)}$ is slower (still less than 0.1 minutes) than the $DP(2,H)$ due to search in clusters and update of the state values on the deterministic shortest path. As disruption level increases, the $CL_{(2,D,U)}$ mitigates the effect of the network size increase better than the $DP(2,H)$ with shorter computational time. Figures 3-(b) show that when the network size increases, the computational time for the hybrid ADP algorithm increases at a much slower rate than the $DP(2,H)$.

"PLACE Fig. 3 ABOUT HERE"

On the other hand, the computational time is mostly affected by the number of disruption levels as the state- and outcome-space increases exponentially with the increase in disruption levels. Fig. 4 shows the computational time of the $CL_{(2,D,U)}$ and the $DP(2,H)$ with respect to different disruption levels. We observe that the $CL_{(2,D,U)}$ computational time lower rate of increase as the number of disruption levels increases as shown in Fig. 4.(a) and (d). The computational time of the $DP(2,H)$ increases at a higher rate when the number of disruption levels

increases. However, as the network size increases with the increase in disruption level the difference between the $CL_{(2,D,U)}$ and the $DP(2,H)$ increases even more. This shows that the $CL_{(2,D,U)}$ mitigates the effect of state- and outcome-space explosion when compared to the $DP(2,H)$.

"PLACE Fig. 4 ABOUT HERE"

5. Conclusions

We have studied a dynamic shortest path problem with travel time-dependent stochastic disruptions. In order to deal with the complexity of the problem, we have proposed a hybrid Approximate Dynamic Programming (ADP) with a deterministic lookahead policy and value function approximation. To further investigate the performance of the proposed algorithm, a test bed of networks with different characteristics are created. In our numerical analyses, we have shown that the hybrid ADP algorithm with the clusters of two- and three-arc-ahead neighborhood significantly outperforms the standard ADP algorithm. The solution quality of hybrid ADP algorithm is higher than or equal to the solution quality of the benchmark heuristic ($DP(2,H)$) when the network size gets larger and the disruption level gets higher. Moreover, the computational time of the hybrid ADP algorithm shows a lower rate of increase with respect to the increase in network size and disruption level. Although $DP(2,H)$ algorithm has relatively good solution quality, for large scale networks with many disruption levels, the hybrid ADP algorithm becomes more attractive with reduced computational time.

Acknowledgements

This research was partially supported by The National Natural Science Foundation of China (NSFC) under Project No. 71771137. The authors would like to thank the Editor-in-Chief and two anonymous reviewers for their helpful comments.

References

Cai, C., Wong, C.K., Heydecker, B.G., 2009. Adaptive traffic signal control using approximate dynamic programming. Transportation Research Part C: Emerging Technologies 17, 456–74.

Demir, E., Huang, Y., Scholts, S., Van Woensel, T., 2015. A selected review on the negative externalities of the freight transportation: Modeling and pricing. Transportation Research Part E: Logistics and Transportation Review 77, 95–114.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische mathematik 1, 269–71.

Eurostat, 2016. Freight transport statistics. Technical Report. Luxemburg: European Commission.

Flatberg, T., Hasle, G., Kloster, O., Nilssen, E.J., Riise, A., 2005. Dynamic and stochastic aspects in vehicle routing-a literature survey. Technical Report. SINTEF Applied Mathematics.

Fu, L., 2001. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. Transportation Research Part B: Methodological 35, 749–65.

Güner, A.R., Murat, A., Chinnam, R.B., 2012. Dynamic routing under recurrent and non-recurrent congestion using real-time its information. Computers & Operations Research 39, 358–73.

Helbing, D., Treiber, M., Kesting, A., Schnhof, M., 2009. Theoretical vs. empirical classification and prediction of congested traffic states. The European Physical Journal B 69, 583–98.

Hulshof, P.J., Mes, M.R., Boucherie, R.J., Hans, E.W., 2016. Patient admission planning using approximate dynamic programming. Flexible Services and Manufacturing Journal 28, 30–61.

Kim, S., Lewis, M.E., White, C., 2005a. Optimal vehicle routing with real-time traffic information. IEEE Transactions on Intelligent Transportation Systems 6, 178–88.

Kim, S., Lewis, M.E., White, C., 2005b. State space reduction for nonstationary stochastic shortest path problems with real-time traffic information. IEEE Transactions on Intelligent Transportation Systems 6, 273–84.

Kumari, S.M., Geethanjali, N., 2010. A survey on shortest path routing algorithms for public transport travel. Global Journal of Computer Science and Technology 9, 73–6.

Lam, S.W., Lee, L.H., Tang, L.C., 2007. An approximate dynamic programming approach for the empty container allocation problem. Transportation Research Part C: Emerging Technologies 15, 265–77.

Medury, A., Madanat, S., 2013. Incorporating network considerations into pavement management systems: A case for approximate dynamic programming. Transportation Research Part C: Emerging Technologies 33, 134–50.

Polychronopoulos, G.H., Tsitsiklis, J.N., 1996. Stochastic shortest path problems with recourse. Networks 27, 133–43.

Powell, W.B., 2011. Approximate dynamic programming: Solving the curses of dimensionality. John Wiley and Sons, Inc., New York. 2nd edition.

Powell, W.B., Shapiro, J.A., Sim£o, H.P., 2002. An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. Transportation Science 36, 231–49.

Powell, W.B., Van Roy, B., 2004. Approximate dynamic programming for high dimensional resource allocation problems. Handbook of learning and approximate dynamic programming , 261–80.

Rehborn, H., Klenov, S.L., Palmer, J., 2011. Common traffic congestion features studied in usa, uk, and germany based on kerner's three-phase traffic theory, in: Intelligent Vehicles Symposium (IV), IEEE. pp. 19–24.

Sever, D., Dellaert, N., Van Woensel, T., de Kok, T., 2013. Dynamic shortest path problems: Hybrid routing policies considering network disruptions. Computers & Operations Research 40, 2852–63.

Simão, H.P., Day, J., George, A.P., Gifford, T., Nienow, J., Powell, W.B., 2009. An approximate dynamic programming algorithm for large-scale fleet management: A case application. Transportation Science 43, 178–97.

Skabardonis, A., Varaiya, P., Petty, K.F., 2003. Measuring recurrent and nonrecurrent traffic congestion. Transportation Research Record: Journal of the Transportation Research Board 1856, 118–24.

Thomas, B., White, C., 2007. The dynamic shortest path problem with anticipation. European Journal of Operational Research 176, 836–54.

Appendix

The paired t-test results and multivariate analysis are given in Table 8 and Table 9, respectively.

"PLACE TABLE 8 ABOUT HERE"

"PLACE TABLE 9 ABOUT HERE"

LIST OF FIGURES

LIST OF TABLES

LIST OF KEYWORDS

Dynamic shortest path problem

Approximate dynamic programming

Time-dependent disruption

Lookahead policy

Value function approximation

Figure 1: The hybrid ADP with clusters

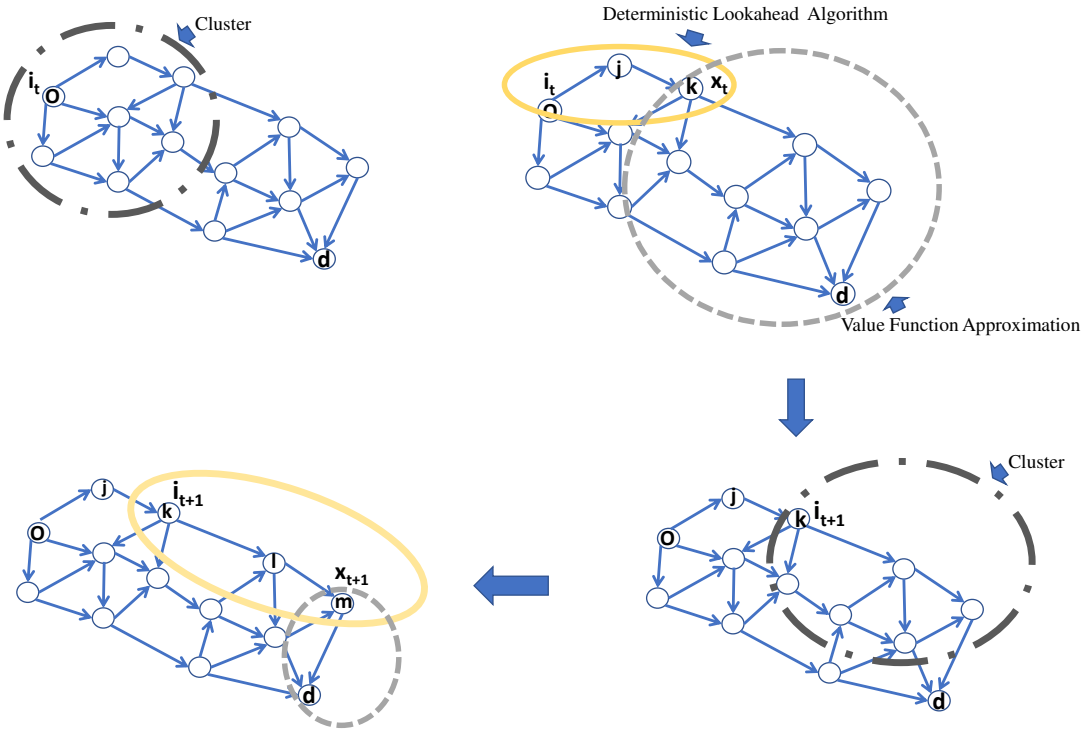
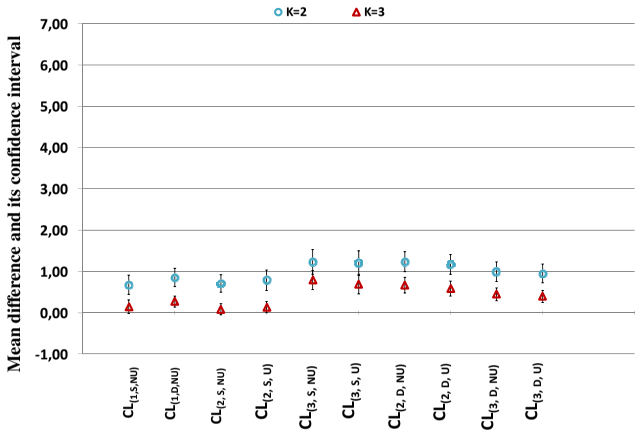


Figure 2: Comparison between initialization heuristics

(a) Small network ($N = 16$)



(b) Large network ($N = 64$)

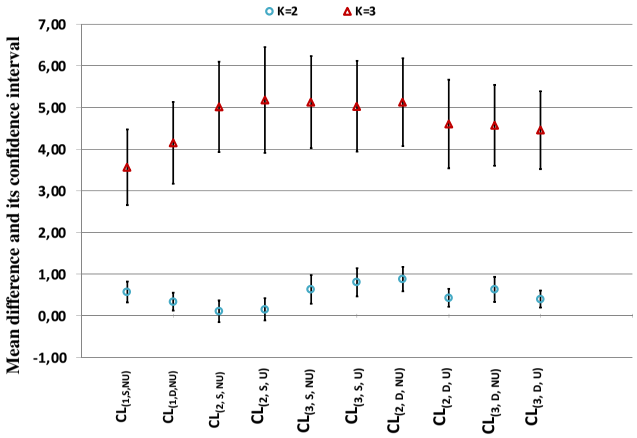
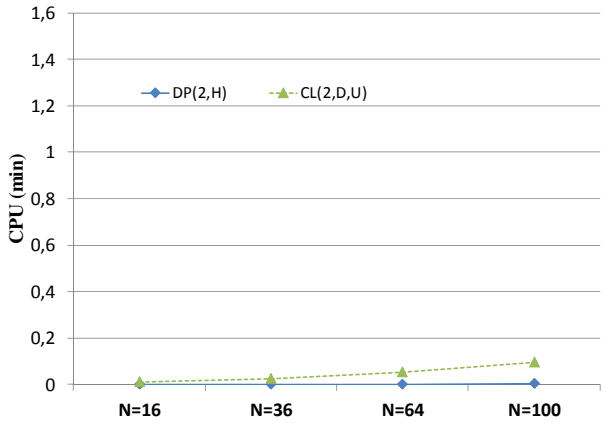
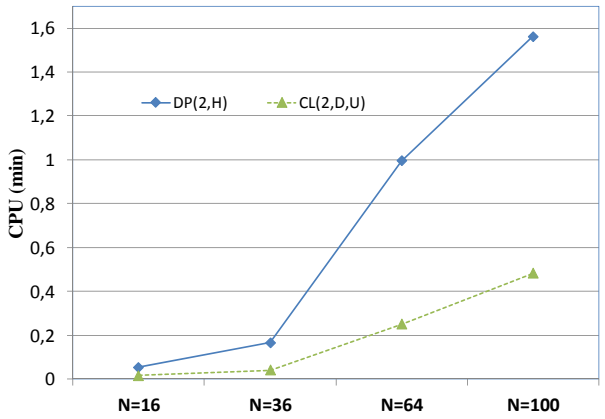


Figure 3: Computational time for different algorithms in instances with different network sizes

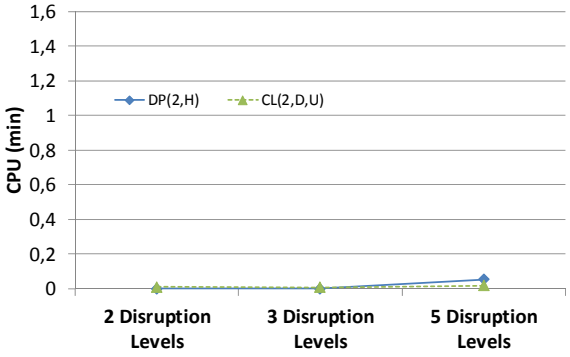


(a) 2-disruption levels

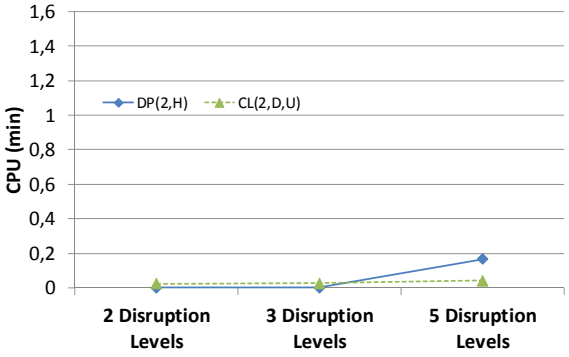


(b) 5-disruption levels

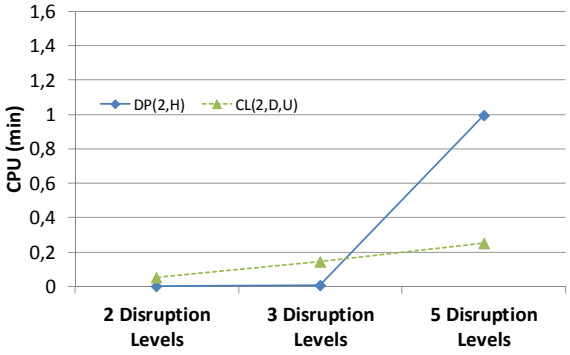
Figure 4: Computational time for different algorithms on instances with different disruption levels



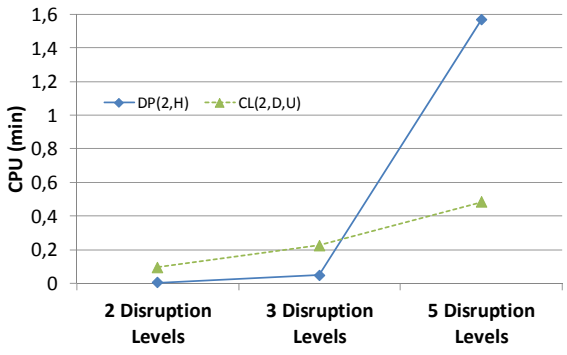
(a) N=16



(b) N=36



(c) N=64



(d) N=100

Table 1: A summary of instance types

Number of nodes	Number of vulnerable arcs		Disruption rate		Number of disruption levels
N	Low	High	Low	High	K
16	3	5	[0.1 - 0.5)	[0.5 - 0.9)	2, 3, 5
36	5	7	[0.1 - 0.5)	[0.5 - 0.9)	2, 3, 5
64	7	9	[0.1 - 0.5)	[0.5 - 0.9)	2, 3, 5*
100*	9	11	[0.1 - 0.5)	[0.5 - 0.9)	2, 3, 5

* In the experiments in Section 4.2 where we analyze several variations of the ADP algorithm, we exclude network instances with $N = 64$ & $K = 5$ and $N = 100$.

Table 2: A summary of algorithmic variations

	Cluster size					
	One-arc-ahead		Two-arc-ahead		Three-arc-ahead	
	No-update	Update	No-update	Update	No-update	Update
Single-pass	$CL_{(1,S,NU)}$	N/A	$CL_{(2,S,NU)}$	$CL_{(2,S,U)}$	$CL_{(3,S,NU)}$	$CL_{(3,S,U)}$
Double-pass	$CL_{(1,D,NU)}$	N/A	$CL_{(2,D,NU)}$	$CL_{(2,D,U)}$	$CL_{(3,D,NU)}$	$CL_{(3,D,U)}$

Table 3: Comparison between the single-pass and the double-pass

	K=2			K=3			K=5		
	Min Δ_{SD}	Max Δ_{SD}	Mean Δ_{SD}	Min Δ_{SD}	Max Δ_{SD}	Δ_{SD}	Min Δ_{SD}	Max Δ_{SD}	Mean Δ_{SD}
$CL_{(1,\cdot,NU)}$	–8.87%	11.16%	**0.54%**	–7.85%	13.34%	**1.22%**	–9.25%	14.41%	**1.30%**
$CL_{(2,\cdot,NU)}$	–3.23%	10.77%	**0.79%**	–3.84%	11.85%	**0.48%**	–9.16%	8.71%	**0.27%**
$CL_{(2,\cdot,U)}$	–3.86%	12.24%	**1.22%**	–3.09%	11.25%	**0.47%**	–4.02%	7.81%	**0.50%**
$CL_{(3,\cdot,NU)}$	–3.48%	12.99%	**1.97%**	–29.28%	11.57%	**1.16%**	–17.51%	10.62%	**1.24%**
$CL_{(3,\cdot,U)}$	–2.96%	18.17%	**3.88%**	–4.02%	20.21%	**2.61%**	–5.16%	14.51%	**2.49%**

(a) Small network (N=16)

	K=2			K=3			K=5		
	Min Δ_{SD}	Max Δ_{SD}	Mean Δ_{SD}	Min Δ_{SD}	Max Δ_{SD}	Δ_{SD}	Min Δ_{SD}	Max Δ_{SD}	Mean Δ_{SD}
$CL_{(1,\cdot,NU)}$	–19.27%	11.94%	–0.37%	–4.59%	18.71%	**1.69%**	–13.94%	13.35%	**0.52%**
$CL_{(2,\cdot,NU)}$	–3.74%	5.36%	**0.55%**	–3.51%	6.02%	**0.27%**	–4.33%	12.05%	**0.06%**
$CL_{(2,\cdot,U)}$	–0.94%	14.33%	**1.93%**	–1.86%	13.95%	**0.32%**	–2.71%	11.88%	**0.21%**
$CL_{(3,\cdot,NU)}$	–1.91%	9.18%	**1.40%**	–4.80%	7.44%	**1.13%**	–2.24%	10.01%	**0.93%**
$CL_{(3,\cdot,U)}$	–2.41%	17.54%	**3.66%**	–2.91%	12.11%	**1.65%**	–3.17%	8.72%	**1.20%**

(b) Medium network (N=36)

	K=2			K=3		
	Min Δ_{SD}	Max Δ_{SD}	Mean Δ_{SD}	Min Δ_{SD}	Max Δ_{SD}	Mean Δ_{SD}
$CL_{(1,\cdot,NU)}$	–10.70%	7.24%	–1.99%	–6.84%	4.16%	–0.90%
$CL_{(2,\cdot,NU)}$	–6.72%	3.91%	**0.06%**	–2.88%	5.87%	**0.25%**
$CL_{(2,\cdot,U)}$	–2.39%	4.26%	**0.73%**	–5.50%	3.85%	**0.10%**
$CL_{(3,\cdot,NU)}$	–5.47%	3.43%	**0.37%**	–2.63%	4.70%	**0.55%**
$CL_{(3,\cdot,U)}$	–2.57%	7.80%	**0.92%**	–4.94%	4.43%	**0.61%**

(c) Large network (N=64)

Table 4: Comparison between no-update and update approaches

	K=2			K=3			K=5		
	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}
$CL_{(2,S,\cdot)}$	–10.08%	4.31%	–0.28%	–6.68%	12.45%	**0.16%**	–15.16%	4.90%	–0.26%
$CL_{(2,D,\cdot)}$	–4.69%	9.42%	**0.18**%	–1.98%	8.38%	**0.15**%	–4.68%	6.52%	**0.02%**
$CL_{(3,S,\cdot)}$	–21.87%	13.77%	–1.94%	–25.85%	6.70%	–1.56%	–26.48%	6.86%	–1.26%
$CL_{(3,D,\cdot)}$	–2.27%	11.70%	**0.19%**	–4.05%	1.41%	**0.03%**	–11.96%	17.78%	**0.08%**

(a) Small network (N=16)

	K=2			K=3			K=5		
	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}
$CL_{(2,S,\cdot)}$	–13.88%	4.15%	–1.11%	–1.97%	9.79%	**0.31%**	–2.48%	3.50%	**0.00%**
$CL_{(2,D,\cdot)}$	–1.08%	4.76%	**0.36**%	–0.88%	5.96%	**0.35**%	–1.56%	3.36%	**0.14%**
$CL_{(3,S,\cdot)}$	–17.88%	6.68%	–2.46%	–11.29%	6.22%	–0.58%	–7.71%	5.72%	–0.27%
$CL_{(3,D,\cdot)}$	–1.44%	2.74%	**0.02%**	–1.15%	2.70%	**0.02%**	–3.31%	1.91%	**0.02%**

(b) Medium network (N=36)

	K=2			K=3		
	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}	Min Δ_{UN}	Max Δ_{UN}	Mean Δ_{UN}
$CL_{(2,S,\cdot)}$	–3.85%	2.98%	–0.05%	–33.04%	25.93%	**0.11%**
$CL_{(2,D,\cdot)}$	–0.64%	5.94%	**0.61**%	–0.69%	4.60%	**0.26**%
$CL_{(3,S,\cdot)}$	–5.32%	2.24%	–0.41%	–3.76%	1.77%	–0.07%
$CL_{(3,D,\cdot)}$	–1.57%	10.17%	**0.15%**	–0.67%	0.74%	**0.01%**

(c) Large network (N=64)

Table 5: Comparison among cluster sizes

	K=2			K=3			K=5		
	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}
$CL_{(1-2,D,U)}$	–6.28%	14.25%	**1.93%**	–2.77%	13.11%	**1.45%**	–11.79%	7.90%	**1.43%**
$CL_{(1-3,D,U)}$	–6.74%	14.97%	**1.74%**	–28.50%	12.78%	**0.95%**	–18.86%	7.85%	**0.77%**

(a) Small network(N=16)

	K=2			K=3			K=5		
	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}
$CL_{(1-2,D,U)}$	–0.54%	9.97%	**3.14%**	–2.20%	11.37%	**3.34%**	–3.50%	13.88%	**4.28%**
$CL_{(1-3,D,U)}$	–1.18%	11.00%	**2.92%**	–4.66%	11.00%	**2.80%**	–5.20%	15.90%	**3.84%**

(b) Medium network (N=36)

	K=2			K=3			K=5		
	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}	Min Δ_{CS}	Max Δ_{CS}	Mean Δ_{CS}
$CL_{(1-2,D,U)}$	–1.17%	13.14%	**5.59%**	1.99%	12.31%	**5.97%**	0.02%	13.27%	**5.74%**
$CL_{(1-3,D,U)}$	–1.81%	15.71%	**4.97%**	1.12%	12.96%	**5.86%**	0.38%	12.45%	**5.73%**

(c) Large network (N=64)

Table 6: The results on different network sizes with low network vulnerability

Network size	Disr. rate	Performance	K=2			K=3			K=5		
			$DP(2,H)$	MDP	$CL_{(2,D,U)}$	$DP(2,H)$	MDP	$CL_{(2,D,U)}$	$DP(2,H)$	MDP	$CL_{(2,D,U)}$
N=16	Low	Cost value	25.37	**25.29**	**25.48**	24.46	24.31	**24.56**	26.72	26.62	**26.78**
		Std.	5.21	**5.23**	**5.23**	5.41	5.40	**5.40**	5.60	5.57	**5.59**
		Min Gap(%)	–	**-5.16%**	**-0.03%**	–	-12.73%	**-0.74%**	–	-10.89%	**-9.99%**
		Max Gap(%)	–	**0.00%**	**4.62%**	–	0.00%	**9.69%**	–	0.00%	**3.83%**
		Mean Gap(%)	–	**-0.29%**	**0.43%**	–	-0.61%	**0.41%**	–	-0.37%	**0.22%**
N=16	High	Cost value	28.22	**28.15**	**28.28**	27.38	27.33	27.46	28.21	28.21	28.37
		Std.	5.32	**5.28**	**5.29**	5.71	5.67	5.59	5.47	5.47	5.48
		Min Gap(%)	–	**-3.81%**	**-3.72%**	–	-2.50%	-0.41%	–	-0.07%	0.00%
		Max Gap(%)	–	**0.00%**	**3.66%**	–	0.00%	3.20%	–	0.00%	4.67%
		Mean Gap(%)	–	**-0.25%**	**0.21%**	–	-0.16%	0.31%	–	0.00%	0.60%
N=36	Low	Cost value	39.48	39.41	39.61	39.59	39.42	39.68	40.20		40.68
		Std.	6.38	6.39	6.36	6.13	6.03	6.21	7.54		7.77
		Min Gap(%)	–	-1.36%	-0.20%	–	-5.80%	-5.48%	–		-2.46%
		Max Gap(%)	–	0.00%	5.05%	–	0.00%	2.69%	–	N/A	4.89%
		Mean Gap(%)	–	-0.17%	0.32%	–	-0.45%	0.21%	–		1.19%
N=36	High	Cost value	42.94	42.90	43.07	41.24	41.19	41.35	42.01		42.34
		Std	6.26	6.27	6.24	6.28	6.17	6.41	5.38		5.73
		Min Gap(%)	–	-1.14%	-0.81%	–	-8.48%	-3.26%	–		-3.15%
		Max Gap(%)	–	0.00%	2.98%	–	0.00%	3.10%	–	N/A	3.69%
		Mean Gap(%)	–	-0.11%	0.30%	–	-0.14%	0.25%	–		0.77%
N=64	Low	Cost value	54.88	**54.46**	**54.69**	53.23		**53.07**	53.46		**53.33**
		Std.	6.84	**6.55**	**6.62**	6.22		**6.07**	6.15		**6.36**
		Min Gap(%)	–	**-3.96%**	**-3.61%**	–		**-6.33%**	–		**-5.48%**
		Max Gap(%)	–	**0.00%**	**4.44%**	–	N/A	**5.33%**	–	N/A	**3.02%**
		Mean Gap(%)	–	**-0.76%**	**-0.33%**	–		**-0.31%**	–		**-0.24%**
N=64	High	Cost value	56.89	56.63	56.90	55.38		55.47	56.30		**56.15**
		Std.	7.31	7.17	6.99	6.15		6.23	6.16		**6.32**
		Min Gap(%)	–	-2.72%	-2.56%	–		-6.14%	–		**-5.37%**
		Max Gap(%)	–	0.00%	4.19%	–	N/A	1.83%	–	N/A	**2.91%**
		Mean Gap(%)	–	-0.46%	0.03%	–		0.17%	–		**-0.27%**
N=100	Low	Cost value	69.02		**68.68**	67.03		66.86	69.28		68.63
		Std.	6.53		**5.75**	6.69		6.61	4.93		4.96
		Min Gap(%)	–		**-8.09%**	–		-4.19%	–		-2.74%
		Max Gap(%)	–	N/A	**3.10%**	–	N/A	3.80%	–	N/A	0.00%
		Mean Gap(%)	–		**-0.48%**	–		-0.26%	–		-0.94%
N=100	High	Cost value	73.01		71.34	69.86		70.08	73.89		73.17
		Std.	7.22		6.31	6.43		6.88	4.58		4.17
		Min Gap(%)	–		-11.28%	–		-0.70%	–		-4.33%
		Max Gap(%)	–	N/A	3.63%	–	N/A	9.57%	–	N/A	1.34%
		Mean Gap(%)	–		-2.29%	–		0.32%	–		-0.99%

Table 7: The results on different network sizes with high network vulnerability

Network size	Disr. rate	Performance	K=2			K=3			K=5	
			$DP(2,H)$	MDP	$CL_{(2,D,U)}$	$DP(2,H)$	MDP	$CL_{(2,D,U)}$	$DP(2,H)$	$CL_{(2,D,U)}$
N=16	Low	Cost value	26.56	**26.38**	**26.65**	**26.70**	**26.38**	**26.81**	28.12	**28.39**
		Std.	5.39	**5.38**	**5.41**	**5.26**	**5.17**	**5.36**	5.95	**6.46**
		Min Gap(%)	–	-5.43%	**-0.55%**	–	**-3.98%**	**-2.85%**	–	**-2.89%**
		Max Gap(%)	–	0.00%	**3.02%**	–	**0.00%**	**8.10%**	–	**14.10%**
		Mean Gap(%)	–	**-0.71%**	**0.33%**	–	**-1.22%**	**0.39%**	–	**0.96%**
N=16	High	Cost value	30.90	30.81	30.81	28.60	28.30	**28.39**	29.45	**29.69**
		Std.	5.51	5.83	5.57	5.66	5.72	**5.79**	5.94	**6.14**
		Min Gap(%)	–	-9.38%	-7.39%	–	-8.89%	**-7.70%**	–	**-2.12%**
		Max Gap(%)	–	0.00%	8.03%	–	0.00%	**2.30%**	–	**5.44%**
		Mean Gap(%)	–	-0.29%	-0.29%	–	-1.05%	**-0.73%**	–	**0.82%**
N=36	Low	Cost value	40.38	40.28	**40.62**	40.30		40.51	41.18	41.41
		Std.	6.34	6.32	**6.34**	6.20		6.21	6.56	6.67
		Min Gap(%)	–	-1.73%	**-0.05%**	–		-3.85%	–	-4.10%
		Max Gap(%)	–	0.00%	**2.37%**	–	N/A	6.15%	–	6.59%
		Mean Gap(%)	–	-0.24%	**0.61%**	–		0.51%	–	0.55%
N=36	High	Cost value	45.12	45.04	45.42	44.03		44.12	44.16	44.31
		Std.	6.65	6.66	6.86	6.10		6.22	5.40	5.33
		Min Gap(%)	–	-0.89%	-0.39%	–		-4.39%	–	-1.76%
		Max Gap(%)	–	0.00%	3.48%	–	N/A	9.23%	–	4.83%
		Mean Gap(%)	–	-0.16%	0.68%	–		0.21%	–	0.34%
N=64	Low	Cost value	57.01		**56.87**	54.21		**53.91**	54.42	54.03
		Std.	6.34		**6.53**	5.61		**5.83**	6.63	6.79
		Min Gap(%)	–		**-7.47%**	–		**-8.54%**	–	-8.98%
		Max Gap(%)	–	N/A	**3.69%**	–	N/A	**1.89%**	–	3.14%
		Mean Gap(%)	–		**-0.25%**	–		**-0.55%**	–	-0.72%
N=64	High	Cost value	59.97		59.52	**57.48**		**57.42**	58.50	58.44
		Std	7.70		7.08	**6.07**		**6.52**	7.48	8.02
		Min Gap(%)	–		-7.73%	–		**-0.68%**	–	-2.85%
		Max Gap(%)	–	N/A	3.56%	–	N/A	**1.61%**	–	3.06%
		Mean Gap(%)	–		-0.74%	–		**-0.11%**	–	-0.09%
N=100	Low	Cost value	71.62		**69.73**	71.20		**70.48**	70.66	69.95
		Std.	6.81		**5.74**	4.51		**4.58**	5.23	4.98
		Min Gap(%)	–		**-9.76%**	–		**-2.82%**	–	-4.12%
		Max Gap(%)	–	N/A	**1.22%**	–	N/A	**-0.18%**	–	0.96%
		Mean Gap(%)	–		**-2.64%**	–		**-1.01%**	–	-1.00%
N=100	High	Cost value	75.02		73.30	73.79		73.61	74.93	74.72
		Std.	8.15		6.33	4.84		4.79	4.79	4.79
		Min Gap(%)	–		-17.21%	–		-1.16%	–	-1.60%
		Max Gap(%)	–	N/A	0.55%	–	N/A	0.83%	–	0.14%
		Mean Gap(%)	–		-2.29%	–		-0.25%	–	-0.29%

Table 8: Paired t-test on initialization heuristics of the ADP algorithm

	Small network				Medium network				Large network			
	K=2		K=3		K=2		K=3		K=2		K=3	
	t	p-value	t	p-value	t	p-value	t	p-value	t	p-value	t	p-value
$CL_{(1,S,NU)}$	5.73	0.00	1.71	0.09	0.88	0.38	7.28	0.00	–4.41	0.00	7.63	0.00
$CL_{(1,D,NU)}$	7.61	0.00	4.05	0.00	2.88	0.00	13.83	0.00	–3.04	0.00	8.26	0.00
$CL_{(2,S,NU)}$	6.62	0.00	1.25	0.21	0.64	0.52	10.35	0.00	0.81	0.42	9.09	0.00
$CL_{(2,S,U)}$	6.33	0.00	1.94	0.05	3.17	0.00	11.12	0.00	1.14	0.26	7.98	0.00
$CL_{(3,S,NU)}$	8.11	0.00	6.76	0.00	5.63	0.00	13.24	0.00	3.61	0.00	9.10	0.00
$CL_{(3,S,U)}$	7.92	0.00	5.61	0.00	3.41	0.00	12.12	0.00	4.68	0.00	9.05	0.00
$CL_{(2,D,NU)}$	9.93	0.00	6.94	0.00	4.88	0.00	12.49	0.00	5.95	0.00	9.52	0.00
$CL_{(2,D,U)}$	9.48	0.00	6.36	0.00	3.71	0.00	11.73	0.00	3.89	0.00	8.46	0.00
$CL_{(3,D,NU)}$	8.24	0.00	5.60	0.00	7.58	0.00	12.50	0.00	4.11	0.00	9.25	0.00
$CL_{(3,D,NU)}$	8.17	0.00	5.31	0.00	5.63	0.00	12.09	0.00	3.89	0.00	9.36	0.00

Table 9: Multivariate analysis on ADP variations

		Small network			Medium network			Large network	
		p-value			p-value			p-value	
Factor	Main Effect-Interaction	K=2	K=3	K=5	K=2	K=3	K=5	K=2	K=3
Main effect	Single-Pass (S) - Double-Pass (D)	0.000	0.000	0.000	0.000	0.000	0.000	0.018	0.045
	No-Update (NU) - Update (U)	0.000	0.000	0.001	0.000	**0.666**	**0.595**	**0.138**	**0.448**
	Cluster size	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Interaction	S-D * NU-U	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.962**
	S-D * Cluster size	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	NU-U * Cluster size	0.000	0.000	0.001	0.000	0.000	**0.105**	0.003	**0.293**
	S-D * NU-U * Cluster size	0.000	0.000	0.000	0.000	0.001	0.002	0.003	**0.293**